CENTRUL DE PREGĂTIRE PENTRU PERFORMANȚĂ HAI LA OLIMPIADĂ !
DISCIPLINA **INFORMATICĂ**
JUDEȚUL **SUCEAVA**

Titlul lectiei: Recursivitatea

Data: 22.10.2016

Profesor: Ștefănescu Daniela Narcisa

Grupa: clasa a X-a **locul de desfasurare**: CN PR Suceava

Recursivitatea

Recursivitatea este un mecanism general de elaborare a algoritmilor. Recursivitatea este proprietatea funcţiilor de a se autoapela. O funcţie se numeşte recursivă dacă ea se autoapelează, fie *direct* (în definiţia ei, se face apel la ea însăşi), fie *indirect* (funcţia X apelează funcţia Y, care apelează funcţia X).

(Un arbore este format din ramuri
O ramură este formată din ramuri mai mici.
O ramură mai mică este formată din ramuri şi mai mici....)

Structura este formată din:

```
tip functie_recursivă (parametru formal)
{ ..
..
..
condiţie de oprire
ramura de continuare
functie_recursivă (parametru formal modificat)
}
```

Recursivitatea a apărut din necesităţi practice date de transcrierea directă a formulelor matematice recursive. Apoi, acest mecanism a fost extins, fiind utilizat în elaborarea multor algoritmi.

Pornim de la un exemplu : Să se calculeze n !

Se observa ca n !=(n-1) !*n şi se stie ca 0 !=1. Aşadar:

5 !=4 !*5
4 !=3 !*4
3 !=2 !*1
1 !=0 !*1
0 !=1 (prin convenţie matematică)

Definiţia recursivă a lui n ! este :

$$n!=\begin{cases}(n-1)!*n, daca & n>0\\ 1 \quad , altfel\end{cases}$$

Functia recursiva se va scrie astfel:

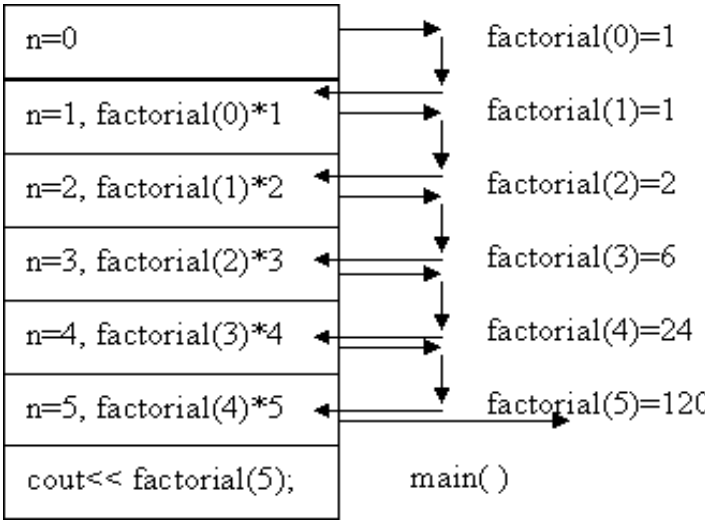
```
long factorial(int n) {
     if(n==0)
         return 1;
     else
         return factorial(n-1)*n;
}

    int main(){
        cout<<factorial(5) ;}
```

Se observă ca funcţia factorial se autoapelează. Autoapelul se realizează prin instructiunea return factorial(n-1)*n.

Să vedem care este mecanismul prin care subprogramele se pot autoapela. Se ştie ca la apelul fiecărui subprogram, se genereaza un nou nivel în segmentul de stivă, corespunzator acelui apel. Pe acel nivel se memorează:

- valorile parametrilor transmişi prin valoare
- adresele parametrilor transmişi prin referinţă
- variabilele locale
- adresa de revenire din funcţie

De fiecare data cand o functie se autoapeleaza, se creeaza un nou nivel în segmentul de stivă.



Fiecare apel de funcţie lucrează cu datele aflate pe nivelul corespunzator acelui apel. La ieşirea din apelul unei funcţii, nivelul respectiv se eliberează şi datele aflate acolo se pierd. Există posibilitatea ca subprogramul să lucreze direct cu variabilele globale, dar în acest caz, subprogramul îşi pierde independenţa.

Cum gândim un algoritm recursiv ?

Iată cateva exemple de rationament recursiv :

- O camera de luat vederi are în obiectiv un televizor care transmite imaginile primite de la cameră. În televizor se vede un televizor în care se vede un televizor…

O gândire recursivă exprimă concentrat o anumita stare, care se repetă la infinit. Această gândire se aplică în elaborarea algoritmilor recursivi cu o modificare esenţiala: adăugarea condiţiei de terminare. În absenţa acestei condiţii nu se poate vorbi despre un algoritm deoarece aceştia sunt finiţi. Din acest punct de vedere, exemplele de mai sus nu sunt corecte.Atunci când gândim un algoritm recursiv, privim problema la un anumit nivel. Ce se întamplă la acel nivel se va întâmpla la toate celelalte.

Observaţii :

În cazul unui număr mare de autoapelări, există posibilitatea ca segmentul de stivă să se ocupe total, caz în care programul se va termina cu eroarea STACK OVERFLOW. Aceasta se întamplă mai ales atunci când condiţia de terminare este pusă greşit şi subprogramul se apelează la nesfârşit.

Pentru orice algoritm recursiv exista unul iterativ care rezolvă aceeaşi problemă.

Mecanismul recursivităţii înlocuieşte instrucţiunile repetitive.

Datorită faptului că la fiecare autoapel se ocupă o zonă de memorie, recursivitatea este eficientă numai dacă numărul de autoapelări nu este prea mare pentru a nu se ajunge la umplerea zonei de memorie alocată.

Recursivitatea oferă avantajul unor soluţii mai clare pentru probleme şi a unei lungimi mai mici a programului. Ea prezintă însă dezavantajul unui timp mai mare de execuţie şi a unui spaţiu de memorie alocată mai mare. Este de preferat ca atunci când programul recursiv poate fi transformat cu uşurinţă într-unul iterativ să se facă apel la cel din urmă (vezi şirul lui Fibonacci)

Exemplu de funcşii ce calculează suma primelor n numere naturale:

Functia iterativa

```
int f(int n)
{int i=1, S=0;
 while  (i<=n)
      {S=S+i ;
       i++ ;}
return S ;}

void main()
{int n=5;
 cout<<f(n);}
```

Functia recursiva 1

```
int n;

int f(int i)
{if  (i<=n)
      return i+f(i+1);
else return 0 ;}

void main()
{n=5;
 cout<<f(1);}
```

Functia recursiva 2

```
int f(int i)
{if  (i>1)
      return i+f(i-1);
else return 0 ;}

void main()
{int n=5;
 cout<<f(n);}
```

Recomandare : înainte de elaborarea algoritmilor recusivi generaţi mai întâi subprogramul iterativ apoi treceţi-l in subprogram recursiv.

- Definiţia numerelor naturale:

  $0 \in N$

  daca $i \in N$, atunci succesorul lui $i \in N$

- Definiţia funcţiei factorial

  fact:N->N

fact(n)=1,dacă n=0
=n*fact(n-1), dacă n>0

- Definiţia funcţiei Fibonacci

fib:N->N
fib(n)=1, dacă n=0 sau n=1
=fib(n-2)+fib(n-1), dacă n>1

Recursivitatea este folosită cu multă eficienţă în matematică. Spre *exemplu*, *definiţii matematice recursive* sunt:

Recursivitatea e strâns legată de iteraţie. *Iteraţia* este execuţia repetată a unei porţiuni de program, până la îndeplinirea unei condiţii (while, do-while, for din C).
Recursivitatea presupune execuţia repetată a unui modul, insa in cursul execuţiei lui (şi nu la sfârşit, ca în cazul iteraţiei), se verifică o condiţie a cărei nesatisfacere implică reluarea execuţiei modulului de la început, fără ca execuţia curentă să se fi terminat. În momentul satisfacerii condiţiei se revine în ordine inversă din lanţul de apeluri, reluându-se şi încheindu-se apelurile suspendate.

Apelul recursiv al unei funcţii trebuie să fie condiţionat de o decizie care să împiedice apelul în cascadă (la infinit); aceasta ar duce la o eroare de program - depăşirea stivei.

Exemplu generic:

```
void p(   ){ //functie recursiva
   p();  //apel infinit
}

//apelul trebuie conditionat in una din variantele:
   if(cond)p();
   while(cond)p();
   do p()while(cond);
```

Tehnica eliminarii recursivitatii

Orice program recursiv poate fi transformat în unul iterativ, dar algoritmul sau poate deveni mai complicat şi mai greu de înţeles. De multe ori, soluţia unei probleme poate fi elaborată mult mai uşor, mai clari şi mai simplu de verificat, printr-un algoritm recursiv.

Se prezinta eliminarea recursivităţii pentru un program simplu, care citeşte caracterele tastate până la un blanc, tipărindu-le apoi în ordine inversă.

```
// varianta recursiva
        void invers_car(void){
```

```
            char car;
            if((car=getche())!=' ')
                invers_car();
            putchar(car);
        }
        void main(void)
            invers_car();
            getche();
        }
// varianta nerecursiva
        void invers_car(void){
            char car;
                                *initializeaza stiva
            while((car=getche())!=' ')
                push(car);
            while(!stiva_goala()){
                pop(car);
                putchar(car);
            }
       }
```

Algoritmi de traversare şi inversare a unei structure de date

Traversarea şi inversarea unei structuri înseamna efectuarea unor operaţii oarecare asupra tuturor elementelor unei structuri în ordine directă, respectiv în ordine inversă. Deşi mai uzuale sunt variantele iterative, caz în care inversarea echivalează cu două traversări directe,variantele recursive sunt mai elegante şi mai concise. Se pot aplica structurilor de tip tablou, listă, fişier şi pot fi o soluţie pentru diverse probleme (transformarea unui întreg dintr-o bază în alta, etc).
Într-o formă generală, algoritmii se pot scrie:

```
    void traversare(tip_element element){  //apelul initial
        // al functiei se face cu primul element al structurii
        prelucrare(element);
        if(element != ultimul_din_structura)
                traversare(element_urmator);
    }     void inversare(tip_element element){  //apelul initial
        // al functiei se face cu primul element al structurii
        if(element != ultimul_din_structura)
                traversare(element_urmator);
        prelucrare(element);
    }
```

Să se calculeze suma elementelor impare din vector.

Fie elementele: 25, 68, 90, 45, 68, 90, 88

Formula matematică:

$S(n) = \{0,\ daca\ n = 0\ S(n-1) + v[n],\ dacă\ n > 0\ si\ v[n]\ par\ S(n-1),\ dacă\ n > 0\ si\ v[n]\ impar\ \}$

Algoritmi care implementeaza definitii recursive

O definiţie recursivă e cea în care un obiect se defineşte prin el însuşi. Definiţia conţine o condiţie de terminare, indicind modul de părăsire a definiţiei şi o parte ce precizează definirea recursivă propriu-zisă.
Ca exemple: algoritmul lui Euclid de aflare a c.m.m.d.c., factorialul, ridicarea la o putere întreagă (prin înmulţiri repetate), definirea recursivă a unei expresii aritmetice, curbele recursive, un mod de a privi permutarile, etc.

Ne vom folosi de algoritmul cmmdc prin împărţiri repetate:

```
  n    m    r
 44 = 20*2 +4
 20 = 4 *5  +0
  5 = 0*
```

```
// functie de determinare a cmmdc cu algoritmul lui Euclid
int cmmdc(int n, int m){
    if(m==0) return n;
    return cmmdc(n,m%n);
}
```

Sau dacă ne oprim pas un pas anterior:

```
// functie de determinare a cmmdc cu algoritmul lui Euclid
int cmmdc(int n, int m){
    if(n%m==0) return m;
    return cmmdc(n,m%n);
}
```

Codul lui Morse. Să se completeze n poziţii cu caracterele *, -. Să se afişeze toate soluţiile posibile.

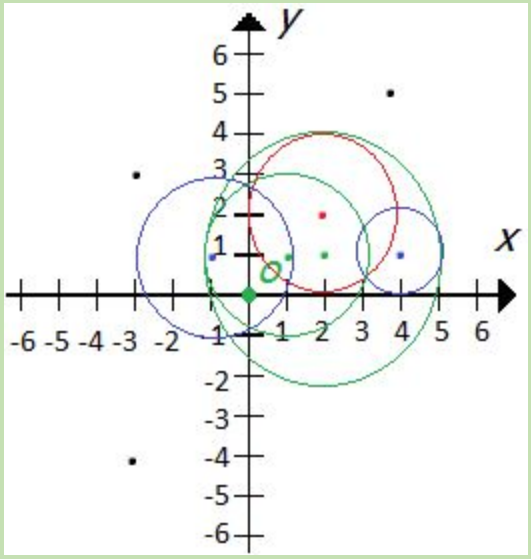
```
void tipar();
void Morse(int pas){
if(pas<n)
   tipar();
else {
    v[pas]='*';
    Morse(pas+1);
    v[pas]='-';
    Morse(pas+1);
}}
```

```
void tipar(){
for(int i=1; i<=n;i++)
   cout<<v[i];
cout<<"\n";
}

int main(){
int n;
cin>>n;
Morse(1);
```

	}

Se dau n bombe, numerotate de la 1 la n, pentru fiecare cunoscându-se coordonatele (x,y) unde sunt plasate şi raza de distrugere r. La explozia unei bombe se va distruge totul în interiorul şi pe cercul de centru (x,y) şi raza r, iar dacă exista alte bombe în această zona, acestea vor exploda la rândul lor. Se dă indicele k al primei bombe care explodează şi se cere să se calculeze câte bombe rămân neexplodate.

Datele se citesc din fişierul bombe.in şi rezultatele se vor afişa în fişierul bombe.out.
În fişierul bombe.in pe prima linie se află numerele n si k, iar pe următoarele n linii coordonatele şi razele de distrugere ale celor n bombe. n si k sunt numere naturale, coordonatele numere întregi, iar razele numere naturale.



Exemplu:
bombe.in
8 5
4 5 4
-3 -4 1
4 1 1
2 1 3
2 2 2
1 1 2
-1 1 2
-3 3 3
bombe.out
3
Explicatie: Prima explodeaza bomba rosie (a 5-a), ea declanseaza cele doua bombe verzi, iar fiecare dintre cele verzi declaseaza cate una albastra. Bombele negre raman neexplodate.

```
#include <fstream>
using namespace std;
ifstream fin("bombe.in");
ofstream fout("bombe.out");

        struct bomba
        {
```

```
        int x,y,p,e;
    };// definim o structura acre sa retina datele pentru o bomba

bomba B[101];
int n,k;

double dist2(int x, int y, int a, int b)
{
  return (x-a)*(x-a)+(y-b)*(y-b);
}//calculam patratul distantei dintre doua puncte din plan

void explozie(int k)
{
  B[k].e=1; // bomba a explodat
  for(int i=1;i<=n;i++)
    if(B[i].e==0 && dist2(B[k].x, B[k].y, B[i].x, B[i].y)<=B[k].p*B[k].p)
      explozie(i); // luam bombele la rand sa vedem daca pot exploda
}

int main()
{
  int c=0;
  fin>>n>>k;
  for(int i=1;i<=n;i++)
    {
      fin>>B[i].x>>B[i].y>>B[i].p;
      B[i].e=0;
    }
  explozie(k);
  for(int i=1;i<=n;i++)
    if(B[i].e==0) c++;
  fout<<c;
  fin.close();
  fout.close();
  return 0;
```

Problema numere - OJI 2003

Problema 2: NUMERE

Gigel este un mare pasionat al cifrelor. Orice moment liber şi-l petrece jucându-se cu numere. Jucându-se astfel, într-o zi a scris pe hârtie 10 numere distincte de câte două cifre şi a observat că printre acestea există două submulțimi disjuncte de sumă egală. Desigur, Gigel a crezut că este o

întâmplare şi a scris alte 10 numere distincte de câte două cifre şi spre surpriza lui, după un timp a găsit din nou două submulțimi disjuncte de sumă egală.

Cerință

Date 10 numere distincte de câte două cifre, determinaţi numărul de perechi de submulţimi **disjuncte** de sumă egală care se pot forma cu numere din cele date, precum şi una dintre aceste perechi pentru care suma numerelor din fiecare dintre cele două submulţimi este maximă.

Date de intrare

Fişierul de intrare **numere.in** conţine pe prima linie 10 numere naturale distincte separate prin câte un spaţiu.

x_1 x_2 ...x_{10}

Date de ieşire

Fişierul de ieşire **numere.out** conţine trei linii. Pe prima linie se află numărul de perechi de submulţimi de sumă egală, precum şi suma maximă obţinută, separate printr-un spaţiu. Pe linia a doua se află elementele primei submulţimi, iar pe linia a treia se află elementele celei de a doua submulţimi, separate prin câte un spaţiu.

NrSol Smax	**NrSol** – numărul de perechi; **Smax** – suma maximă
x_1 ... x_k	elementele primei submulţimi
y_1 ... y_p	elementele celei de a doua submulţimi

Restricții şi precizări

- $10 \le x_i, y_i \le 99$, pentru $1 \le i \le 10$
- $1 \le k, p \le 9$
- Ordinea submulţimilor în perechi nu contează.
- Perechea de submulţimi determinată nu este obligatoriu unică.

Exemplu

numere.in	**numere.out**	Semnificaţie
60 49 86 78 23 97 69 71 32 10	130 276 78 97 69 32 60 49 86 71 10	**130** de soluţii; suma maximă este **276**, s-au folosit **9** din cele **10** numere prima submulţime are **4** elemente, a doua are **5** elemente

Timp maxim de executare/test: 1 secundă

Completam un vector care sa caracterizeze numerele date:

0 – nu apartine la nicio multime,

1 – la multimea A,

2- la B.

//Varianta lui Edy, mai optimizata decat ce am discutat la clasa.

```
#include <bits/stdc++.h>
using namespace std;
ifstream fin("numere.in");
ofstream fout("numere.out");
int v[11],a[11],num,M=-10e9,Ma[11];
void gen(int p,int s1, int s2){
    if(p==11){
        if(s1==s2&&s1!=0)
        {
            num++;
            if(s1>M)
            {
                M=s1;
                for(int i=1;i<=10;i++)
                    Ma[i]=a[i];
            }
        }
    }else{
    a[p]=0;
    gen(p+1,s1,s2);
    a[p]=1;
    gen(p+1,s1+v[p],s2);
    a[p]=2;
    gen(p+1,s1,s2+v[p]);
    }
}
int main()
{
    for(int i=1;i<=10;i++)
        fin>>v[i];
    gen(1,0,0);
    fout<<num<<" "<<M<<"\n";
    for(int k=1;k<=2;k++){
        for(int i=1;i<=10;i++)
            if(Ma[i]==k)
            fout<<v[i]<<" ";
        fout<<"\n";
    }
    return 0;
}
```

Aveti aici metoda comisiei:

```
#include <fstream.h>
#include <stdlib.h>

int a[11], s1max[11], s2max[11], solutii=0, Smax=0, UzMax=0;


ifstream f("numere.in");
ofstream g("numere.out");

void citire()
{
  int i;
  for (i=1; i<=10; i++) f>>a[i];
  f.close();
}

void rezolva()
{
  int  vi[11]={0,0,0,0,0,0,0,0,0,0,0}, vj[11]={0,0,0,0,0,0,0,0,0,0,0};
  int i, ii, j, k, x, gasit, si, sj, jj, disjuncti;
  int cati1;

  for(ii=1; ii<=1022; ii++)
    {
      i=10;
      while(vi[i]) vi[i--]=0;      //adun 1 la vi
      vi[i]=1;

      si=0;
      for (i=1; i<=10; i++)
         if (vi[i]) si+=a[i];
      for (i=1; i<=10; i++) vj[i]=vi[i];
      for (jj=ii+1; jj<=1022; jj++)
       {
         j=10;
         while(vj[j]) vj[j--]=0;  //adun 1 la vj
         vj[j]=1;

         sj=0;
         for(j=1; j<=10; j++)
           if(vj[j]) sj+=a[j];

         if(si==sj)
           {
             disjuncti=1;
             for(k=1; k<=10; k++)
              if((vi[k]+vj[k])==2)
                disjuncti=0;
             if(disjuncti)
              {
                solutii++;
                cati1=0;
                for (i=1; i<=10; i++) cati1+=(vi[i]+vj[i]);
                if (cati1>UzMax)
                  if (si>Smax)
```

```
                    {
                     UzMax=cati1; Smax=si;
                     for (i=1; i<=10; i++) s1max[i]=vi[i];
                     for (i=1; i<=10; i++) s2max[i]=vj[i];
                    }
               }
            }
         }
      }
}

void afisare()
{
  int i;
  g<<solutii<<" "<<Smax<<endl;
  for(i=1; i<=10; i++)
    if (s1max[i]) g<<a[i]<<' ';
  g<<endl;
  for(i=1; i<=10; i++)
    if (s2max[i]) g<<a[i]<<' ';
  g<<endl;
  g.close();
}

void main()
{
  citire();
  rezolva();
  afisare();
}
```